Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Goran Ferbišek

Prepoznava obrazov na mobilnih platformah

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Borut Batagelj

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Sistemi za prepoznavo obrazov so že zelo uspešni in razširjeni, kljub temu pa jih na mobilnih napravah ne najdemo veliko. Preglejte področje prepoznave obrazov in različnih rešitev na mobilnih napravah. Preverite katere metode so primerne za implementacijo na mobilnih napravah in izbrano metodo tudi preizkusite in implementirajte. Aplikacija naj bo zabavne narave, ki na podlagi obraza določi najbolj podobnega igralca izbrane nogometne ekipe.

Zahvaljujem se mentorju viš. pred. dr. Borutu Batagelju, za pomoč in nasvete pri izdelavi diplomske naloge. Zahvaljujem se tudi družini za podporo in spodbudo pri celotnem študiju.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ADT	Android Development Tools	razvojna orodja Android
PCA	Principal Component Analysis	Metoda glavnih komponent
LDA	Linear Discriminant Analysis	Linearna diskriminantna analiza
XML	Extensible Markup Language	Razširljiv označevalni jezik
JVM	Java Virtual Machine	Java virtualni stroj
API	Application Programming Interface	Aplikacijski programski vmesnik
USB	Universal Serial Bus	univerzalno serijsko vodilo
SVD	Singular Value Decomposition	razcep s singularnimi vrednostmi
LBP	Local Binary Patterns	Lokalni binarni vzorci
CNN	Convolutional Neural Network	konvolucijska nevronska mreža

Povzetek

Naslov: Prepoznava obrazov na mobilnih platformah

Avtor: Goran Ferbišek

Področje prepoznave obrazov se največ ukvarja s programi, ki so namenjeni varnostnim sistemom, malo pa je aplikacij namenjenih zabavi. Večina teh programov je napisanih za namizne računalnike. Zato smo se odločili izdelati aplikacijo zabavne narave, ki deluje na mobilnem operacijskem sistemu Android. Aplikacija je namenjena primerjavi z znanimi nogometnimi igralci. Uporabnik usmeri kamero pametnega telefona tako, da ujame svoj ali pa prijateljev obraz. Aplikacija prikaže kateremu znanemu nogometašu v bazi slik je zajeti obraz najbolj podoben. Naredili smo pregled obstoječih aplikacij za prepoznavo obrazov na mobilnih platformah. Pregledali smo različne metode za prepoznavo obrazov, med njimi pa smo opisali implementacijo metode glavnih komponent. Algoritem za prepoznavo obrazov smo testirali na bazi obraznih slik FERET. Opisali smo postopek za pripravo obrazov na prepoznavo, tako da dajejo najboljše rezultate.

Ključne besede: prepoznava obrazov, metoda glavnih komponent, Android, openCV, FERET.

Abstract

Title: Face recognition on mobile platforms

Author: Goran Ferbišek

The field of face recognition is dealing mostly with security system applications. Only a small portion of these applications are made with entertainment in mind. Most of the software on the market is written for personal computers. This was our motivation to make an entertainment application for Android mobile operating system. The application use is intended for comparing persons faces to celebrities. In our case, we chose football players. The user directs the smart phone's camera towards a persons face and captures it. As a result, the application shows which face in the database, it resembles the most. We made an overview of similar applications currently available. We also described several face recognition methods. Our implementation of Principal Component Analysis was described in detail. Our method was tested on FERET database of facial images. We described the procedure of image preparation for best results possible.

Keywords: face recognition, Principal Component Analysis, Android, openCV, FERET.

Poglavje 1

Uvod

S prepoznavo obrazov se ukvarjata področji računalniškega vida in umetne inteligence. Uporabljata metodo prepoznave različnih vzorcev. Pri računalniškem vidu se izvaja pridobivanje, obdelava in analiziranje digitalnih slik. V splošnem gre za izločanje vzorcev iz visoko-dimenzionalnih podatkov, v obliki fotografije, videa (tudi več kamer hkrati) ali pa podatki medicinskih čitalnikov (angl. *medical scanner*). Skupaj z umetno inteligenco se na področju računalniškega vida ukvarjajo z zaznavo dogodkov, zaznavo gibanja, sledenje objektom, prepoznavo teksta in govora, ter odločitvenimi sistemi.

V tej diplomski nalogi se bomo osredotočili na prepoznavo obrazov. Želimo si, da bi aplikacija najprej obraz prepoznala. Za učinkovito prepoznavo obraza ga moramo najprej primerno obdelati. Koraki obdelave vključujejo poravnavo obraza, primerno pomanjšanje velikosti, svetlobno normalizacijo.

Program, izdelan v okviru diplomske naloge je zasnovan tako, da uporabnik usmeri kamero telefona v obraz osebe, aplikacija pa zazna obraz. Ko aplikacija obraz zazna, lahko uporabnik sproži zajem slike. Aplikacija iz zajete slike pridobi obraz, ki ga nato obdela s prej naštetimi postopki. Zadnji korak je primerjava zajetega obraza z obrazi, ki jih imamo shranjene v bazi na mobilnem telefonu. Kot rezultat aplikacija prikaže obraz kateremu je zajeti obraz najbolj podoben.

Na trgu obstajajo tudi komercialne aplikacije za uporabo na spletu, na-

miznih računalnikih in na mobilnih platformah. Več o aplikacijah za mobilne platforme smo zapisali v poglavju 1.1, tu pa bomo opisali nekaj aplikacij, za ostale platforme, ki so motivirale pri izdelavi aplikacije razvite v tej diplomski nalogi.

Prepoznava obrazov se pogosto uporablja v komercialnih aplikacijah, kot so urejevalniki foto albumov. Primer je nekdanji program Picassa [41], ki ga je zamenjala storitev Google Photos [21], ter program Photos [40] podjetja Apple.

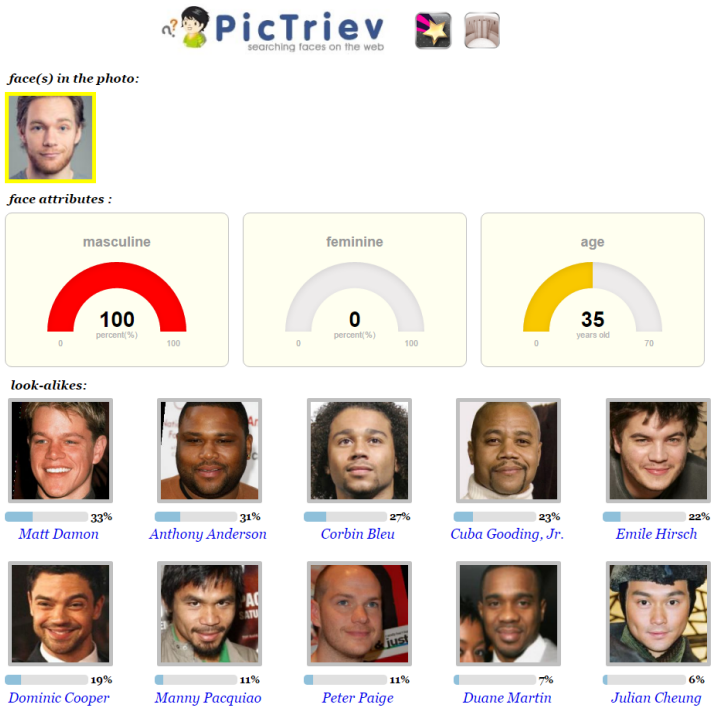
Prepoznavo obrazov uporabljajo tudi urejevalniki slik kot sta Adobe Photoshop Lightroom [2] in Adobe Photoshop Elements [1].

Prepoznava obrazov se uporablja tudi v komercialne namene, tako da aplikacije prepoznavajo katere osebe se na slikah pojavljajo skupaj, kar pomeni, da imajo nekaj skupnega. Pogosto so to socialna omrežja kot sta Facebook [18] in Google+ [20].

Spletni iskalnik slik PicTriev [42] je po namenu uporabe zelo podoben aplikaciji v tej diplomski nalogi. Na spletno stran naložimo sliko osebe, vrne pa seznam znanih oseb, ki so razvrščene po podobnosti. Uporabniški vmesnik je viden na sliki 1.1. Hkrati pove tudi oceno, koliko smo podobni moškemu ali ženski, pove pa tudi koliko je oseba stara. Ugotavljanje spola in starosti so sorodna področja zaznave obrazov, nekatere aplikacije pa skušajo ugotoviti tudi naša čustva.

1.1 Pregled mobilnih aplikacij za prepoznavo obrazov

Na mobilnih platformah obstajajo aplikacije, ki ponujajo varnostne rešitve. Njihov namen je preprečevanje organiziranega kriminala, iskanje pogrešanih oseb in preprečevanje vstopa neavtoriziranim osebam. Uporablja se lahko v vojski, policiji, letališčih, trgovinah, stadionih itd. Primer takšne aplikacije, ki deluje tudi na operacijskem sistemu Android je aplikacija FaceFirst [43]. Podobna je tudi aplikacija FaceTec [44], ki deluje tudi na operacijskem sis-



Slika 1.1: Spletni iskalnik PicTriev

temu iOS.

Na pametnih telefonih se uporablja prepoznava obrazov tudi za odklepanje zaslona. Android jo ponuja od verzije Android 4.0 Ice Cream Sandwich dalje. Enako deluje tudi aplikacija Windows Hello [17] na operacijskem sistemu Windows 10 Mobile.

V storitvi Google play [24] najdemo še nekaj aplikacij, ki uporabljajo prepoznavo obrazov. Aplikacija Luxand Face Recognition [33] demonstrira rezultate razvojnega paketa FaceSDK, ki ga razvija podjetje Luxand. Aplikacija Face Recognition [15] razvijalca SeakLeng uporablja knjižnico OpenCV, enako kot aplikacija razvita za potrebe te diplomske naloge. Istoimenska aplikacija razvijalca sladomic [16] je zanimiva zaradi uporabe nevronskih mrež kot eno izmed metod prepoznave obrazov, ki so trenutno vedno bolj zanimiva tema. Aplikacija ponuja veliko nastavitev. Izbiramo lahko velikost zajetega obraza, zaznavo obraza z OpenCV ali pa z razredom Android FaceDetector.

Aplikacija za prepoznavo zajame več obrazov zaporedoma.

Na trgovini z aplikacijami Google Play [24] smo poiskali aplikacije s podobnim ciljem. Vsako smo namestili na svoj pametni telefon in opravili preizkus. Aplikacija Celebrity Look Alike App Free [23] je namenjena primerjavi obraza z znanimi osebami. Izbiramo lahko s katero skupino oseb se bomo primerjali. Vključeni so komiki, foto modeli, igralci, najbogatejše osebe in junaki. Aplikacija na videz vrača zelo naključne zadetke, kar je znak da prepoznava ni uspešna. Ocena na trgovini Google Play je zelo slaba, kar je potrjeno tudi z negativnimi komentarji.

Aplikacija My Lookalike [25] oglašuje storitev, da v svoji bazi oseb poišče dvojčke ali dvojnike oseb (angl. *doppelganger*). Bazo obrazov gradijo uporabniki sami tako, da prispevajo svoje slike. Komentarji in ocene so negativne. Aplikacija nudi dobro zaznavo obraza in oči. Za prepoznavo je potrebno ustvariti uporabniški račun s osebnimi podatki, kar pa med našim testom ni uspelo.

Preizkusili smo tudi aplikacijo PicFace Celebrity Matchup [22]. Omogoča primerjavo obrazov z znanimi osebami, ima pa tudi funkcijo beleženja zgodovine vseh primerjav. Baza znanih oseb se nahaja na strežniku, saj se aplikacija pogosto povezuje nanj. Komentarji in ocene so slabe. Ob našem testu nam je večkrat javila napako na omrežju ali pa informacijo, da ujemanje obrazov ni bilo uspešno.

Najbolje se je izkazala aplikacija My Twin Celeb [26]. Od vseh naštetih je bila edina pozitivno ocenjena s strani uporabnikov. Uporabniku omogoča, da sliko zajame s kamero ali pa izbere predhodno zajeto fotografijo iz galerije mobilnega telefona. Izpiše tudi visoko podobnost med osebama, vizualno pa bi ocenili, da rezultat ni zadovoljiv. Ob večkratnem poizkusu je rezultat pogosto enak, kar pomeni, da ne vrača naključnih zadetkov.

Prepoznava obrazov v teh aplikacijah je neuspešna, kar je še dodatna motivacija k izdelavi lastne. Večina teh aplikacij je namenjena oglaševanju, saj se nam med uporabo prikazujejo oglasi. Nekatere zahtevajo tudi vpis osebnih podatkov.

Poglavje 2

Programska oprema in tehnologije

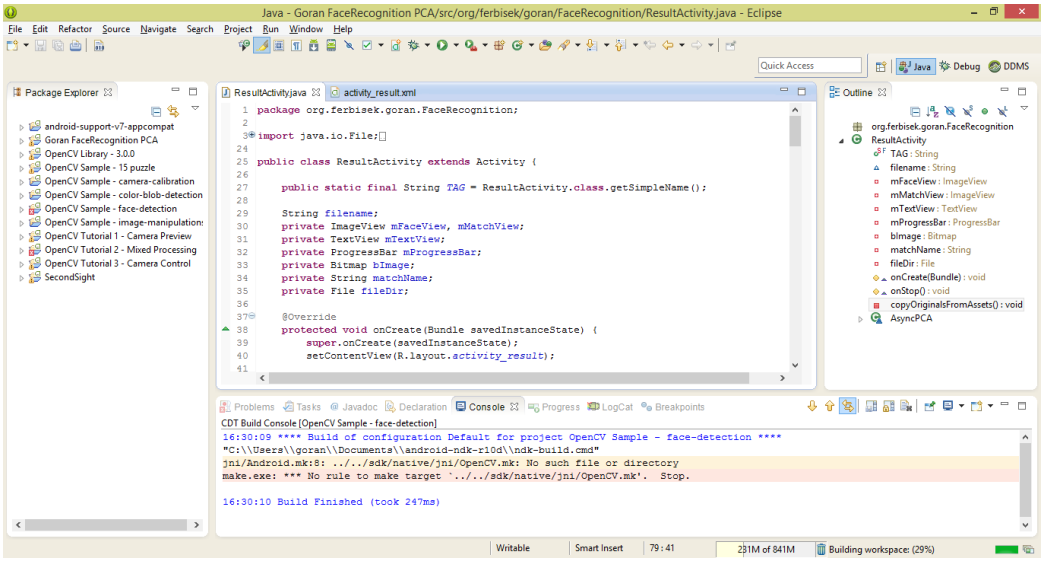
2.1 Programska oprema

2.1.1 Razvojno okolje Eclipse

Programska oprema Eclipse je integrirano razvojno okolje namenjeno razvoju programov v programskem jeziku Java. Z namestitvijo vtičnikov je možno razvijati programe tudi v drugih popularnih programskih jezikih, kot so: Ada, C, C++, COBOL, Fortran, Haskell, JavaScript, Perl, PHP, Prolog, Python, R, in Ruby.

Vtičnik ADT omogoča razvoj aplikacij za operacijski sistem Android. Z vtičnikom ADT lahko hitro ustvarimo ogrodje novega projekta za Android aplikacijo. Vključuje orodje za vizualno urejanje uporabniškega vmesnika, orodje za razhroščevanje, in orodje za izvoz aplikacij v obliko za nadaljnjo distribucijo.

Podjetje Google je prenehalo z razvojem vtičnika ADT. Novo uradno razvojno okolje se imenuje Android Studio, ki temelji na razvojnem okolju IntelliJ IDEA.



Slika 2.1: Uporabniški vmesnik razvojnega orodja Eclipse z vtičnikom ADT

2.2 Tehnologije

2.2.1 Java

Java [29] je objektno usmerjeni programski jezik. Razvil ga je James Gosling s sodelavci v podjetju Sun Microsystems. Kasneje ga prevzelo podjetje Oracle. Razvit je bil kot zamenjava za programski jezik C++, skladnjo pa povzema po programskem jeziku C.

Programski jezik Java je prenosljiv. Programsko kodo prevedemo v vmesno bitno kodo (angl. *bytecode*). To kodo lahko brez ponovnega prevajanja poženemo na vseh platformah, ki jih podpira programski jezik Java. Je popolnoma neodvisna od računalniške arhitekture. Na ciljni platformi potrebujemo le JVM (angl. *Java Virtual Machine*).

Programski jezik Java se uporablja za razvoj aplikacij za osebne računalnike, mobilne naprave, ter za spletne aplikacije, kjer teče na strani strežnika. Je tudi glavni jezik za razvoj aplikacij na mobilnem operacijskem sistemu Android.

2.2.2 Android

Mobilni operacijski sistem Android [5] razvija podjetje Google. Prvotno je namenjen za pametne mobilne telefone, sedaj pa se širi tudi na televizijske sprejemnike, avtomobile in pametne ure. Osnova operacijskega sistema Android je odprto kodni operacijski sistem Linux. Večji del je napisan v programskem jeziku Java, delno pa tudi v jezikih C in C++.

Na operacijskem sistemu Android se programska koda napisana v Javi ne izvaja na virtualnem stroju JVM. Do verzije sistema Android 4.4 KitKat se koda izvaja na virtualnem stroju Dalvik. Od različice Android 5.0 Lollipop naprej ga nadomešča ART (Android Runtime) [11]. Prednost ART je boljše upravljanje s pomnilnikom in izboljšana učinkovitost izvajanja kode, saj kodo pred izvajanjem dodatno optimizira.

Za razvoj aplikacij potrebujemo razvojno okolje oziroma IDE (angl. *Integrated development environment*). Na voljo sta razvojni okolji Eclipse, ki ni več uradno podprta [12], ter Android Studio. Obe razvojni okolji vsebujeta urejevalnik kode s funkcijo opozarjanja na napake, orodje za vizualno načrtovanje uporabniškega vmesnika, orodje za gradnjo (angl. *build tools*), razhroščevalnik (angl. *debugger*), ter emulator za poganjanje in testiranje aplikacij v virtualni napravi. Najbolje je da pri testiranju aplikacije poganjamo na dejanskih napravah preko USB kabla. Predhodno moramo na napravi nastaviti razhroščevalni način (angl. *debugg mode*).

Aplikacije za Android lahko razvijalci naložijo na trgovino Google Play. Trgovina vsebuje večinoma brezplačne aplikacije lahko pa tudi plačljive. Uporabniki si s povezavo na Google Play lahko enostavno prenesejo in namestijo aplikacije na svoj pametni telefon, tablico ali katero koli napravo na kateri teče Android.

2.2.3 XML

Označevalni jezik XML [49] (angl. *Extensible Markup Language*) sestavlja množico pravil za tekstovno kodiranje podatkov. Namenjen je predvsem za

enostavnejšo izmenjavo velikih količin podatkov. Besedilo je strukturirano tako, da je berljivo za ljudi in za računalnike.

Za oblikovanje poljubne strukture podatkov lahko sami definiramo svoje značke (angl. *tags*). Značka je konstrukt, ki se zančne z znakom `<` in konča z znakom `>`. Poznamo začetno značko `<LinearLayout>` in končno značko `</LinearLayout>`. Poznamo tudi značko za prazen element `<ImageView />`. Značka lahko vsebuje tudi enega ali več atributov s katerim podamo lastnosti elementom, ki smo jih definirali. Atribute definiramo takoj za imenom začetne značke `android:padding="16dp"`.

Jezik XML se uporablja v tehnologijah kot so: XHTML, RSS, SOAP, Twitter Markup Language, Office Open XML, OpenDocument.

2.2.4 Knjižnica OpenCV

OpenCV je odprtokodna knjižnica za računalniški vid. Izdana je pod BSD licenco, zato je uporabna tako za akademsko in za komercialno uporabo. V osnovi je napisana v programskih jezikih C in C++, na voljo pa so tudi vmesniki za druge programske jezike. Med njimi so: Python, Java in MATLAB. Podprta je na sledečih operacijskih sistemih: Windows, Linux, Android, iOS in Mac OS.

Žal v Javi niso implementirane vse funkcije, na operacijskem sistemu Android pa niso na voljo vsi moduli. Tudi dokumentacija je pomanjkljivo napisana ali pa je kopirana iz dokumentacije za programski jezik C++.

Poglavje 3

Metode prepoznave obrazov

Na področju računalništva oziroma računalniškega vida govorimo o prepoznavi obrazov takrat, ko na digitalni sliki ali video posnetku najprej zaznamo obraz, nato pa ga primerjamo z obrazi, ki jih imamo vnaprej obdelane in shranjene v podatkovni bazi. Uporabimo lahko cel obraz ali pa samo določene značilnosti kot so usta, nos in oči. Tipično se uporablja v varnostnih sistemih za prepoznavanje sumljivih ali želenih oseb. Sorodna področja se ukvarjajo s prepoznavo tudi ostali biometričnih podatkov kot so prstni odtis, šarenica, hoja, podpis in govor.

3.1 Metode osnovane na videzu

Pri metodah osnovanih na videzu je vhodni podatek slika celega obraza. To lahko predstavlja težavo, saj takšna metoda zahteva veliko virov, tako pomnilnik kot tudi procesorsko moč za obdelavo. Zato se praviloma za prepoznavo uporablja sivinske slike. S tem zmanjšamo velikost slik za tretjino, saj uporabimo samo 8 bitov za predstavitev posamezne slikovne točke. Za predstavitev v RGB prostoru jih uporabimo trikrat toliko.

Za zmanjšanje dimenzije podatkov uporabljamo sledeče metode: PCA, LDA in ICA. Te metode so podrobneje opisane v tem poglavju. Z njimi analiziramo obraz in ga pretvorimo v vektorje manjših dimenzij, ki jim pravimo

lastni vektorji obraza.

Težavo predstavljajo tudi spremenljivi pogoji ob zajemu obraza. Na kakovost slike vplivajo različne kamere, osvetlitev, izraz na obrazu.

3.1.1 Metoda glavnih komponent

Pri prepoznavi obrazov s pomočjo metode glavnih komponent [10, 13] (angl. *Principal Component Analysis*, PCA) lahko več-dimenzionalen podatek, kot je slika, preslikamo v nižje-dimenzijski prostor. Metoda glavnih komponent zavrti koordinatni sistem tako, da lahko raznolikost podatkov predstavimo že z majhnim številom vektorjev, ki jih imenujemo glavne komponente. Metoda mora zagotoviti, da je varianca podatkov čim večja. Komponente so urejene od najpomembnejše do najmanj pomembne. To pomeni, da glavna komponenta opisuje največjo razpršenost vhodnih podatkov. Cilj je izbrati nekaj začetnih komponent, ki pojasnjujejo večji del razpršenosti. Tako s čim manjšo izgubo informacij zmanjšamo dimenzije podatkov. V poglavju 4.3 podrobneje opišemo implementacijo metode.

3.1.2 Linearna diskriminantna analiza

Cilj metode linearna diskriminantna analiza [10, 31] (angl. *Linear discriminant analysis*, LDA) je razpršitev podatkov na razrede. V našem primeru so razredi različne osebe. Metoda mora poiskati matriko, ki določa razporeditev podatkov med razredi. Med razredi je zaželena čim večja razpršenost. Metoda poišče tudi matriko, ki določa razporeditev znotraj posameznih razredov, kjer mora biti razpršenost podatkov čim manjša. Lastni obrazi so poimenovani Fisherjevi obrazi (angl. *fisherfaces*).

V primerjavi z metodo PCA je manj občutljiva na različne svetlobne pogoje. Metoda LDA se zelo dobro prilagodi učnim primerom. V primeru, ko je bila slika posneta v drugačnih pogojih, se uspešnost metode zmanjša.

3.1.3 Metoda neodvisnih komponent

Z metodo neodvisnih komponent [10, 28] (angl. *Independent Component Analysis*,ICS) sprejmemo signal, nato pa ga razčlenimo v kombinacijo neznanih neodvisnih signalov. Z njo največkrat ločujemo signale mešanih izvorov (angl. *Blind Source Separation*, BSS). Zato se pogosto uporablja analizo različnih neodvisnih signalov (npr. video ali govor). Metoda sprejme vektor izvornih signalov, najde pa takšno mešalno (angl. *mixing matrix*) oziroma ločevalno matriko, da bodo signali čimbolj neodvisni. Metoda je zelo podobna metodi glavnih komponent.

3.2 Metode osnovane na značilkah

Tudi pri metodah osnovanih na značilkah najprej poiščemo dele obraza kot so nos, oči in usta. Z njimi izvedemo poravnavo obraza, kot je opisano v poglavju 4.2. Sliko potem razdelimo na pod-slike, iz katerih izračunamo značilke. Za prepoznavo se uporabijo lokacija, ter geometrijske in vizualne lastnosti. Na obrazih primerjamo različne dimenzije in razdalje med posameznimi značilkami obraza.

3.2.1 LBP

Z metodo lokalnih binarnih vzorcev [4] (angl. *Local Binary Pattern*, LBP) izvajamo prepoznavo obrazov tako, da slikovne točke na posamezni sliki primerjamo s sosednjimi. Tam kjer je sredinska točka svetlejša (vrednost točke je večja) od sosednje zapišemo 1, sicer pa 0. Vsaka točka ima osem sosednjih. Preberemo jih v smeri urinega kazalca, pri čemer dobimo 8-bitno dvojiško število. Števila nato pretvorimo v desetiški številski sistem in iz njih izračunamo histogram. Histogrami vsebujejo informacijo o lokalnih značilkah kot so robovi, točke ali ravna območja. Za klasifikacijo obrazov, med histogrami računamo razdalje. Metodo lahko tudi prilagajamo s povečevanjem radija okoli posamezne točke. Ena izmed možnih optimizacij je šteje spre-

memb iz 0 v 1 ali obratno, v njihovem zaporedju.

3.2.2 Metoda Gaborjevih valčkov

Pri metodi Gaborjevih valčkov [10, 19] (angl. *Gabor wavelets*) za predstavitev posameznih slik uporabimo tako imenovane valčke. Slike primerjamo tako, da posamezno sliko predstavimo z Gaborjevimi valčki v prej definiranih točkah na sliki. Običajno točke enakomerno razporedimo po sliki. Sliko razdelimo na mrežo, kjer v vsaki točki izračunamo množico Gaborjevih filtrov [19]. Le ti poudarjajo glavne značilke na obrazu kot so oči, nos, usta, ter nepravilnosti na obrazu, ki imajo izrazito obliko. Zato se pogosto uporablja na področju zaznavanja robov in prepoznavo črk (angl. Optical character recognition, OCR). Zaradi svojih značilnosti je metoda podobna Fourier-jevi analizi.

3.3 Nevronske mreže

Konvolucijske nevronske mreže (angl. *Convolutional Neural Networks*, CNN) se pri obdelavi informacij zgledujejo po bioloških mrežah. Oponašajo delovanje živčnih povezav v možganih, ki so zelo prilagodljive, sposobne pa so rešiti probleme, ki jih ni mogoče formulirati kot algoritem. Delujejo na podlagi iskanja vzorcev. Nevronske mreže [30] so sestavljene iz več nivojev nevronov. Pri vhodnem nivoju je število nevronov enako številu učnih slik, izhodni nivo pa predstavlja posamezne klasifikacijske razrede.

Da so nevronske mreže zelo učinkovite pri prepoznavi sta dokazali tudi veliki podjetji kot sta Google in Facebook. Sistem Google FaceNet [39] je dosegel 99,63% točnost, sistem Facebook DeepFace [48] pa 97,35% točnost na slikovni bazi podatkov Labeled Faces in the Wild, ki zajema 13 tisoč slik obrazov. Podjetje Google je svoj sistem testiralo tudi na slikovni bazi YouTube Faces Database, kjer so dosegli 95.12% točnost.

Pri sistemu Google FaceNet so vzeli med sto in dvesto milijonov učnih slik, ki zajemajo 8 milijonov različnih oseb. Tako zgrajena nevronska mreža zajema 140 milijonov parametrov. Učenje na tako veliki množici podatkov,

je na gruči procesnih enot, trajalo med tisoč in dva tisoč ur.

Podjetje Facebook je za sistem DeepFace uporabilo 4 milijone učnih slik, ki zajemajo 4 tisoč oseb. Njihova nevronska mreža zajema 120 milijonov parametrov.

Najbolj znana nevronska mreža, naučena na obrazih, se imenuje VGG-Face [38]. Nevronsko mrežo so zgradili na 2,6 milijona slik, med njimi je bilo 2622 unikatnih oseb. Na večini slik so bile znane osebe, saj so tako lahko našli velike količine slik različnih oseb, s potrjeno identiteto. Takšne slike so tudi prosto dostopne na svetovnem spletu. Za primerjavo so na bazi slik Labeled Faces in the Wild dosegli 99,13% točnost, na bazi slik YouTube Faces Dataset pa 97,4% točnost.

Poglavje 4

Razvoj aplikacije

V sklopu diplomske naloge smo izdelali aplikacijo za mobilni operacijski sistem Android. Namenjena je primerjavi podobnosti med nami in znanimi osebami. V tem primeru smo izbrali nogometne igralce. Aplikacija je zabavne narave in ne služi natančni prepoznavi obraza, kot je to običajno v varnostnih sistemih.

Aplikacija je v smislu uporabniškega vmesnika precej enostavna. Sestavljena je samo iz dveh aktivnosti (angl. *Android Activity*). Android Activity [6] predstavlja enoto uporabniškega vmesnika za interakcijo z uporabnikom. Praviloma je to okno, ki se razteza čez cel zaslon.

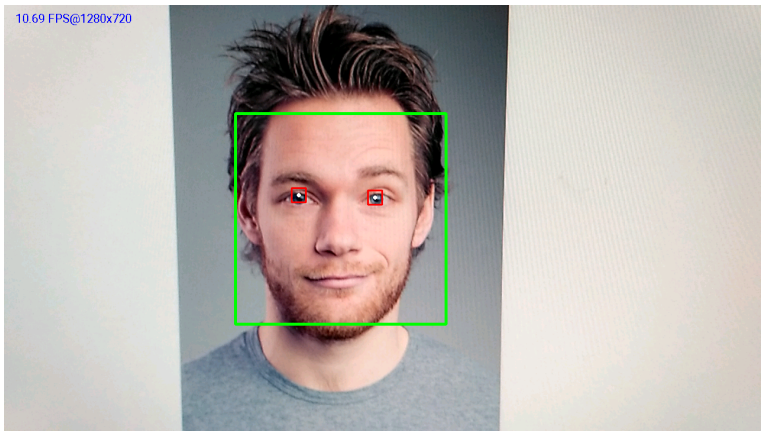
Za izdelavo aplikacije smo uporabili razvojno okolje Eclipse z vtičnikom ADT, čeprav je bilo v času izdelave le to že uradno zamenjano z okoljem Android Studio. Eclipse IDE smo uporabili zato, ker je bila povsod v literaturi [27] prikazana uporaba knjižnice OpenCV z orodjem Eclipse. Enako tudi na uradni spletni strani projekta OpenCV [35].

Za uporabo knjižnice OpenCV smo se odločili zaradi težav z metodami za zaznavo obrazov, ki so že implementirane v programski vmesnik za Android [46]. Razred *android.hardware.Camera.Parameters*, ki implementira funkcijo *getMaxNumDetectedFaces()*, testira sposobnost naprave za prepoznavo obrazov tako, da vrne število obrazov, ki jih lahko odkrije. Ugotovili smo, da na veliko različnih napravah ta metoda vrača vrednost 0, kar pomeni

da zaznave obrazov ne podpira.

Običajno vključevanje zunanjih programskih knjižnic zahteva več prostora na trdem disku, oziroma se poveča velikost same aplikacije. Lahko imamo na enem telefonu več aplikacij, ki izkoriščajo funkcionalnost te knjižnice. OpenCV ima ta problem rešen tako, da ponuja aplikacijo, ki se namesti kot storitev [7] (angl. *service*), ne kot klasična aplikacija za sistem Android. Aplikacija se imenuje OpenCV Manager [37]. Ob namestitvi naše aplikacije telefon sam ponudi prenos aplikacije iz trgovine Google Play [24].

4.1 Detekcija obraza



Slika 4.1: Zaznavanje obraza z razredom MainActivity.java

Začetni zaslon aplikacije predstavlja razred *MainActivitiy.java*. Skrbi za zagon kamere in prikazovanje zajete slike na zaslon. Velika večina pametnih telefonov na trgu ima kamero spredaj in zadaj. Za namen te aplikacije bi bila najbolj primerna uporaba sprednje kamere, saj želimo da uporabnik zajame svoj obraz. Tu smo naleteli na omejitev v knjižnici OpenCV, ki ne deluje pravilno na sprednji kameri. Zato smo aplikacijo zasnovali tako, da zažene kamero na zadnji strani telefona. Zajem s sprednje kamere sicer deluje, vendar samo v ležečem načinu (angl. *landscape mode*). Tudi tu se pojavi težava

s popačenostjo slike, saj je sprednja kamera namenjena uporabi v pokončnem načinu (angl. *portrait mode*). Napako opažajo tudi ponekod v literaturi [27] in na spletnem sistemu za prijavo napak OpenCV DevZone [36]. Ena možna rešitev je zajem slike v ležečem načinu, nato pa jo uporabniku prikažemo rotirano za 90 stopinj. Pri tem se pojavi zatikanje sličic v videu, saj naprava opravlja preveč računskega dela v metodi za zajem slik *onCameraFrame()*.

4.1.1 Klasifikatorja Haar in LBP

Za detekcijo obraza s knjižnico OpenCV poznamo dve metodi. Poznamo detekcijo s kaskadnim klasifikatorjem Haar [32] (angl. *Haar Cascade Classifier*) ali pa klasifikator LBP [45] (angl. *Local Binary Patterns*).

Klasifikator Haar deluje na principu objektov, ki primerjajo temna in svetla področja na sliki. Razvila sta jo avtorja Viola in Jones. Običajno so lica na obrazu svetlejša, usta, čelo in nos pa temnejša. Deluje za iskanje tako obrazov, kot posameznih značilk obraza kot so: nos, oči, usta. Pred potrditvijo, da je metoda zaznala obraz, naredi okoli 20 različnih preverjan. To mora narediti za vsako možno pozicijo na sliki in hkrati za vsako možno velikost obraza. Za eno sliko naredi na tisoče preverjanj. Metoda je sicer počasnejša, vendar dosega 95 odstotno točnost [9] pri odkritju obraza.

Metoda s klasifikatorjem LBP deluje na podoben način. Razvita je bila iz strani avtorjev Ahonen, Hadid in Pietikäinen. Metoda LBP deluje na principu primerjave jakosti svetlobe sosednjih slikovnih točk. Metoda je sicer manj natančna, vendar je nekajkrat hitrejša [9].

Knjižnica OpenCV ima oba klasifikatorja že naučena iz ogromne množice primerov. Rezultati so shranjeni v XML datoteki, za kasnejšo uporabo. Uporabljenih je vsaj 1000 obrazov (pozitivnih primerov) in 10000 ne obrazov (negativnih primerov). Vsebuje klasifikatorje za zaznavo obrazov, oči, ust, ter tudi obrazov iz stranskega profila.

Uporabili smo klasifikator LBP, saj deluje hitreje, kar je pomembno na pametnih telefonih, ki imajo omejene vire v primerjavi z namiznimi računalniki. XML datoteke so shranjene v mapi */res/raw*, ki se nahaja v strukturi An-

droid projekta. Datoteko moramo prebrati z razredom *CascadeClassifier*.

```
private CascadeClassifier      mFaceDetector;
File mCascadeFile = new File(R.raw.lbpcascade_frontalface,
                             "lbpcascade_frontalface.xml");
```

Za učinkovito prepoznavo moramo obraz najprej ustrezno pripraviti. Za ta namen potrebujemo tudi pozicijo oči. Te smo zaznali z uporabo klasifikatorja Haar, ki ga naložimo iz datoteke *haarcascade_lefteye_2splits.xml*.

Najprej na sliki zaznamo obraz z funkcijo *detectMultiscale()*.

```
mJavaDetector.detectMultiScale(mGray, faces, 1.1, 2,
                               Objdetect.CASCADE_SCALE_IMAGE,
                               new Size(mAbsoluteFaceSize,
                                   mAbsoluteFaceSize),
                               new Size());
```

Z isto funkcijo najdemo še oči, vendar iščemo samo znotraj najdenega obraza.

4.2 Priprava obraza - normalizacija

Testni obraz moramo pred prepoznavo ustrezno obdelati. Enako velja za vse slike v galeriji. Vsi obrazi morajo biti enako poravnani. Oči morajo biti vedno na enakem mestu. Zato potrebujemo pozicijo oči, da lahko obraz ustrezno rotiramo. S tem poskrbimo da so oči vodoravno poravnane. Sliko potem zmanjšamo, tako da je razdalja med očmi vedno enaka. Na koncu sliko obrežemo na neko vnaprej določeno velikost. Tako na sliki ostanejo samo elementi pomembni za prepoznavo. Poskrbeti moramo tudi za izenačitev različne osvetlitve slik, saj se lahko uporabnik s pametnim telefonom nahaja vedno v različnih okoljih in pogojih.

Galerijo slik je potrebno nekako vstaviti v projekt. V okolju Android ni dovoljeno ustvarjati map v drevesni strukturi, razen podmap znotraj mape

/assets in */res/drawable*. Slednja lahko vsebuje samo grafične datoteke, beremo pa jih lahko samo preko unikatnih identifikatorjev, ki so enaki imenu datoteke. Zato ne moremo dobiti seznama fotografij, ki so shranjene v mapi */res/drawable*. Odločili smo se, da galerijo obrazov shranimo v mapo */assets*, od koder lahko dobimo seznam datotek.

```
AssetManager am = getAssets();
try {
    assetFiles = am.list(getString(R.string.assets_face_folder));
} catch (IOException e) {
    Log.w(TAG, "Error reading from assets: ", e);
}
```

Prebrane datoteke smo shranili na zunanji pomnilnik, ki je običajno notranji pomnilnik telefona ali pa spominska kartica (angl. *micro SD card*). V mapo */originals* smo shranili barvne slike za prikaz in nadaljnjo obdelavo. Po obdelavi opisani v tem poglavju pa smo jih shranili v mapo */cropped* za prepoznavo.

4.2.1 Rotacija slike obraza

Obraz moramo rotirati tako, da sta obe očesi horizontalno poravnani. Za rotacijo potrebujemo kot med njima. Za vsako oko poznamo koordinati x in y. Med njima izračunamo vertikalno in horizontalno razliko. Kot izračunamo po sledeči formuli [10]:

$$\alpha = \frac{V_{raz}}{H_{raz}} \tag{4.1}$$

Če je kot pozitiven, sliko rotiramo v nasprotni smeri urinega kazalca. To se zgodi kadar je levo oko višje od desnega. Drugače rotiramo v nasprotni smeri.

```
private double eyeAngle(Point left, Point right) {
    double heightDiff = left.y - right.y;
```

```
    double widthDiff = left.x - right.x;
    return Math.atan(heightDiff/widthDiff);
}
```

Napisali smo funkcijo, ki s pomočjo funkcij knjižnice OpenCV izvede rotacijo slike. Potrebujemo sredinsko točko slike in kot funkcije. Te parametre podamo funkciji *Imgproc.getRotationMatrix()*. Tretji parameter je faktor za povečavo ali zmanjšanje slike, ki ga zaenkrat ne potrebujemo, zato je faktor nastavljen na vrednost 1. To pomeni, da se velikost slike ne spremeni. Funkcija nam vrne rotacijsko matriko, ki jo potem podamo funkciji *Imgproc.warpAffine()*, ki sliko ustrezno zasuka [3].

```
private Mat rotateFace(Mat src, double angle) {
    Mat dst = new Mat();
    Point center = new Point(image.width()/2, image.height()/2);
    Mat r = Imgproc.getRotationMatrix2D(center,
                                Math.toDegrees(angle), 1);
    Imgproc.warpAffine(src, dst, r, src.size());
    return dst;
}
```

Potem moramo popraviti tudi pozicijo oči [10], ki jo potrebujemo v naslednjem koraku. Zato smo implementirali funkcijo *translateEyePosition()*.

```
private Point translateEyePosition(Point eye) {
    //premaknem koordinatni sistem
    double x = eye.x - image.cols()/2;
    double y = image.rows()/2 - eye.y;

    double rx = Math.cos(-angle) * x + Math.sin(-angle)*y;
    double ry = -Math.sin(-angle) * x + Math.cos(-angle)*y;

    //popravimo koordinatni sistem nazaj
```

```
    double nx = rx + image.cols()/2;
    double ny = image.rows()/2 - ry;

    return new Point(nx,ny);
}
```

4.2.2 Sprememba velikosti

Razdalja med očmi in njihova pozicija mora biti na vseh slikah enaka. Za spremembo velikosti slike potrebujemo faktor povečave [10]. Izračunamo ga tako, da delimo želeno razdaljo s trenutno. Za pomanjšanje smo uporabili funkcijo *Imgproc.resize()*. Naše slike so praviloma manjše od zajetih, saj bi pri povečavi zmanjšali kvaliteto slik. Faktor je zato manjši od nič ($ratio < 0$).

```
double eyeDistance = Math.sqrt( Math.pow(LX-RX,2) +
                                Math.pow((LY-RY),2));
double ratio = d0/eyeDistance;
Imgproc.resize(image, image, new Size(),
               ratio, ratio, Imgproc.INTER_AREA);
```

Po spremembi velikosti slike se ponovno spremenijo koordinate oči. Nove dobimo tako, da jih pomnožimo s faktorjem povečave.

4.2.3 Rezanje slike obraza

Vse slike smo obrezali na enotno velikost 87x140 slikovnih točk. S poskušanjem smo ugotovili, da je to najboljša velikost, saj se s povečevanjem slik uspešnost prepoznave ni povečala. Z obrezovanjem odstranimo večino ozadja, ki niso del obraza. Hkrati zagotovimo tudi vedno enako pozicijo obraznih značilk. Določili smo vertikalno razdaljo med očmi in zgornjim robom slike, tako da so oči na 45% višine slike. Definirali smo tudi odmik od levega očesa, tako da so oči na sredini. Tako dobimo levi zgornji rob obraza, ki ga bomo izrezali iz fotografije. Horizontalna razdalja med očmi je 48 slikovnih

točk. Pri tako izbranih parametrih dobimo levo oko na koordinatah (19,63) za levo oko in (67,63) za desno oko. Izhodišče koordinatnega sistema je v zgornjem levem kotu.

```
int heightNorm = 140;
int widthNorm = 87;
double xRez = LX - 19;
double yRez = LY - 63;
this.image = this.image.submat((int)yRez,
                               (int)yRez+heightNorm,
                               (int)xRez,
                               (int)xRez+widthNorm);
```

4.2.4 Svetlobna normalizacija

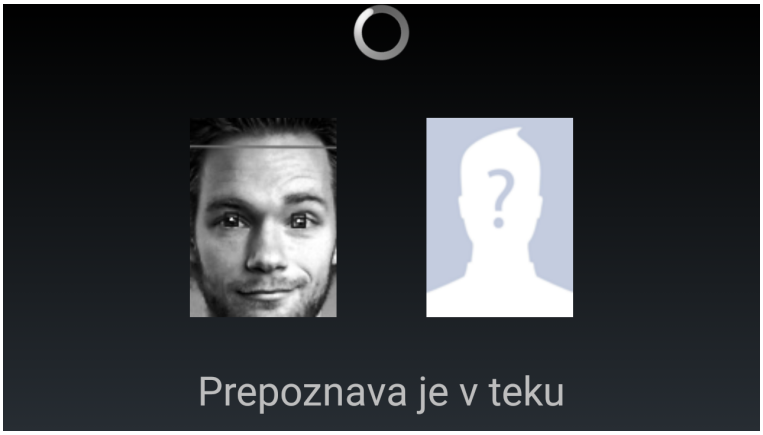
Slike smo svetlobno normalizirali z metodo izravnave histograma. Implementacija te metode ni bila potrebna, saj je že implementirana v knjižnici OpenCV. Metoda poskrbi za enakomerno porazdelitev različnih odtenkov sive. Podati ji moramo sivinsko sliko.

```
Imgproc.equalizeHist(image, image);
```

4.3 Prepoznava obraza - PCA metoda

Prepoznavo obrazov smo implementirali v razredu *ResultActivity.java*. Na zaslonu se prikaže testni obraz, ki smo ga zajeli s kamero. Zaradi računske zahtevnosti smo uporabili razred *AsyncTask* [8] v Android API, s katerim smo ustvarili novo nit (angl. *thread*). S tem poskrbimo, da glavna nit aplikacije, ki skrbi za izris uporabniškega vmesnika ne zamuja pri prikazu. Med izračunom uporabniku prikažemo indikator (angl. *progress bar*) zato, da uporabnik nima občutka o zablokirani aplikaciji. Uporabniški vmesnik je prikazan na sliki 4.2. Končni rezultat aplikacije, ko najdemo najbolj podoben obraz v galeriji je prikazan na sliki 4.5.

Za implementacijo smo izbrali metodo glavnih komponent (angl. *Principal Component Analysis*, PCA). Za implementacijo tega algoritma potrebujemo operacije na matrikah. Standardna knjižnica programskega jezika Java žal tega ne omogoča. Programski vmesnik Android API (angl. *Application Programming Interface*) ima razred *android.graphics.Matrix*, vendar vsebuje predvsem funkcije za operacije na slikah, nima pa matematičnih operacij kot je množenje matrik. Za ta namen smo našli zunanjo neodvisno knjižnico *Apache Commons Math* [34], ki smo jo vključili v projekt v obliki datoteke JAR (angl. *Java ARchive*).



Slika 4.2: Obdelava obrazov z metodo PCA v razredu ResultActivity.java

4.3.1 Postopek učenja z metodo PCA

1. Vhod v metodo je galerija obrazov, katere bomo primerjali s testnim obrazom, ki ga posname uporabnik aplikacije. Galerija slik mora biti predhodno obdelana, kot je opisano v prejšnjem poglavju. Postopek učenja se izvede samo na začetku, ko vnesemo slike v galerijo. Izračunani vektorji se shranijo za kasnejšo primerjavo. Galerijo slik vstavimo v matriko X, pri čemer so stolpci matrike vektorji, ki ima $m \times n$ elementov, kolikor je slikovnih točk posamezne slike. Pri dimenzijah slike 87×140 to pomeni 12180 elementov. Dolžino posameznega

vektorja bomo poimenovali M. Slike so sivinske, kar pomeni, da vsak element pove intenziteto vsake točke. Število stolpcev matrike X je enako številu učnih primerov. To število poimenujemo N.

2. Izračunamo povprečni obraz tako, da izračunamo povprečno vrednost enako ležečih slikovnih točk. To pomeni, da izračunamo povprečje vsake vrstice matrike X.

Slika 4.3: Povprečen obraz matrike X

3. Povprečni vektor odštejemo od stolpcev matrike X. Tako dobimo normalizirane obraze.

4. Za izračun lastnih obrazov [14, 10] (angl. *eingenfaces*) in lastnih vrednosti (angl. *eigenvalues*) moramo izračunati kovariančno matriko dobljene matrike X. Želimo da je med vsemi vhodnimi slikami čim večja varianca. Kovariančno matriko izračunamo tako da pomnožimo matriko X z njeno transponirano matriko.

$$XX^T \tag{4.2}$$

Težava je v tem, da je za velikost vektorja M kovariančna matrika velikosti $M \times M$. V našem primeru (M=12180) je rezultat matrika dimenzije 12180×12180. To pomeni, da že pri razmeroma majhnih

slikah (87x140 slikovnih točk) postane računsko prezahteven problem. Dokazano je, da lahko uporabimo matriko notranjega produkta [10].

$$X^T X \tag{4.3}$$

V tem primeru je velikost matrike $N \times N$, kjer je število učnih primerov (N=100) mnogo manjše od dimenzije vektorjev. Učni primeri so v našem primeru galerija slik znanih oseb. Tako se računska zahtevnost močno zmanjša.

5. Lastne obraze in lastne vrednosti izračunamo s pomočjo razcepa s singularnimi vrednostmi [47] (angl. *Singular Value Decomposition*, SVD). Za razcep smo uporabili razred *SingularValueDecomposition* implementiran v programski knjižnici *Apache Commons Math*.

```
SingularValueDecomposition svd =
         new SingularValueDecomposition(covarianceMatrix);
RealMatrix u = svd.getU(); // u - lastni vektorji
```

6. Izkaže se, da vseh lastnih obrazov ne potrebujemo, saj nimajo vsi enake pomembnosti. Začetni nosijo večji del informacij, zadnji pa predstavljajo šum pri zaznavi obrazov, kot je vidno na sliki 4.4. Urejeni so po pomembnosti, zato vzamemo samo začetno množico. Tako še dodatno zmanjšamo računsko zahtevnost aplikacije, saj množimo manjše matrike.

```
public final static int NUM_OF_EIGENFACES = 100;
...
int eigenfacesCount = Math.min(NUM_OF_EIGENFACES,
                               v.getDimension());
RealMatrix eigenMat = u.getSubMatrix(0,
                                 gallery.gallerySize()-1,
```

```
                                    0,
                                    eigenfacesCount-1);
```

Slika 4.4: Prvih 20 lastni obrazov

7. S pomočjo lastnih obrazov projiciramo posamezno sliko. S tem določimo transformacijsko matriko W. Stolpce dobljene matrike moramo normalizirati.

```
RealMatrix w = subtractedFaceMat.multiply(eigenMat);
w = normColums(w);  // normalizira stolpce matrike
```

8. S pomočjo transformacijske matrike projiciramo vse slike iz galerije v obrazni prostor.

```
RealMatrix pGallery = w.transpose().multiply(subtractedFaceMat);
```

Tako izračunano projekcijo vseh slik shranimo v telefon na pomnilni medij poleg slik. Shranimo tudi povprečno vrednost in transformacijsko matriko.

4.3.2 Postopek prepoznave z metodo PCA

1. Preberemo testno sliko zajeto iz kamere v enodimenzionalno matriko $X1$, od nje pa odštejemo povprečni vektor, ki smo ga izračunali na učnih primerih.

2. Testni obraz s pomočjo transformacijske matrike W preslikamo v obrazni prostor.

   ```
   RealMatrix x1 = testFaceMat.transpose().multiply(w);
   ```

3. Na koncu izračunamo razdaljo vsakega obraza iz galerije z vhodnim obrazom. Za razdaljo smo uporabili razred *EuclideanDistance*, knjižnice Apache Commons Math.

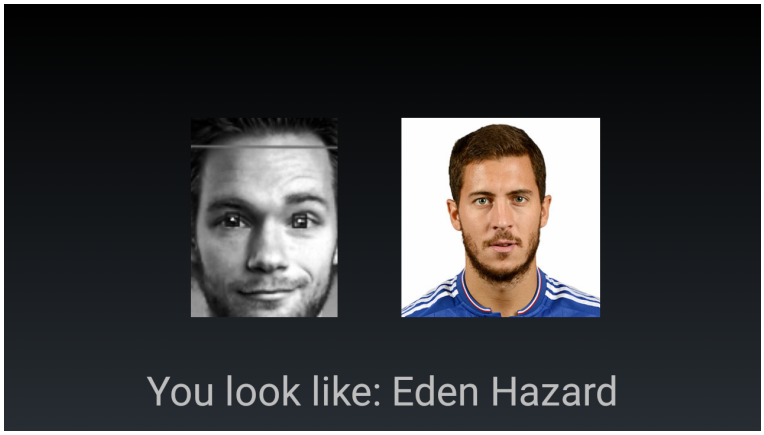
   ```
   double[] razdalje = new double[gallery.gallerySize()];
   EuclideanDistance d = new EuclideanDistance();

   for (int i = 0; i < razdalje.length; i++) {
       razdalje[i] = d.compute(y2.getRow(0),
                     pGallery.getColumn(i));
   }
   ```

   Indeks najbolj podobne slike v tabeli najdemo tako, da najdemo najmanjšo razdaljo.

4.4 Preizkus uspešnosti - Baza FERET

Aplikacijo smo preizkusili na bazi obraznih slik FERET. To je standardna baza slik za ocenjevanje uspešnosti sistemov za prepoznavo obrazov. Vsebuje



Slika 4.5: Najden obraz v razredu ResultActivity.java

več množic slik, vse pa so sivinske v velikosti 256x384 slikovnih točk. Množice lahko vsebujejo slike z obrazi obrnjenimi proti kameri, iz levega ali desnega profila, različnimi izrazi na obrazu, ter z različnimi osvetlitvami.

Kot primarno galerijo smo izbrali množico obrazov, kjer so obrazi z različnimi izrazi: nasmejani, žalostni, resni, jezni itd. Testirali smo na množici *fafb*, kjer so iste osebe vendar imajo drugačen izraz na obrazu.

V obeh množicah so bili obrazi poravnani, tako da so oči vedno na istem mestu. Vse slike so sivinske, predhodno pa smo jih pomanjšali na velikost 87x140 slikovnih točk. Vsako sliko iz testne množice smo primerjali, s sliko iz primarne galerije. Med dvema slikama smo nato izračunali razdaljo. Za izračun razdalje je bila izbrana evklidska razdalja. Učnih slik smo imeli 1196. Na množici *fafb*, ki vsebuje 1195 slik smo dosegli 90,04% uspešnost.

Poglavje 5

Sklepne ugotovitve

V diplomskem delu smo razvili aplikacijo za operacijski sistem Android. Aplikacija prepozna obraz uporabnika in ga zajame. Potem obraz ustrezno obdela tako, da je pripravljen na prepoznavo. Obraz uporabnika primerja z bazo slik nogometnih igralcev. Kot rezultat, nam aplikacija pove, komu smo najbolj podobni. Aplikacija je sicer zabavne narave. Ni namenjena visoki točnosti prepoznave, saj v našem primeru iskane osebe ni v bazi. Iščemo samo najbolj podobno osebo.

V implementaciji imamo še prostor za izboljšave. Iz rezultatov v poglavju 4.4 vidimo, da so ti močno odvisni od kakovosti vhodnih slik, saj smo dobili zadovoljiv rezultat le pri testni množici *fafb*, kjer ima oseba samo drugačen izraz na obraz.

Metoda za prepoznavo se je slabo odrezala na množici *fafc*, kjer je drugačna osvetlitev, ki ima velik vpliv na prepoznavo. To je ena izmed večjih pomanjkljivosti metode glavnih komponent. Dokazano je, da to težavo rešimo, če zavržemo nekaj začetnih lastnih obrazov [10]. Ti nosijo največ informacije o osvetlitvi. V nadaljnjem delu bi lahko implementirali še kakšno od metod omenjenih v poglavju 3.

Običajno sta leva in desna stran obraza drugače osvetljena. Upoštevati moramo tudi, da se lahko uporabnik pametnega telefona nahaja vedno v drugačnih pogojih. Izboljšavo bi lahko izvedli tudi pri izravnavi histograma

obraza tako, da posebej naredimo izravnavo leve in desne strani [9]. Zato da se izognemo grobemu prehodu leve in desne strani, naredimo izravnavo še na sredini slike.

Rezultat bi lahko izboljšali z uporabo druge razdalje. Boljša izbira bi bila kosinusna razdalja. Še bolje pa bi bilo upoštevati lastne vrednosti (angl. *eigenvalues*), kot uteži pri razdaljah. Ena izmed takšnih razdalj je razdalja Mahalanobis.

Literatura

[1] Photoshop Elements — Photoshop.com. Dosegljivo: http://www.photoshop.com/products/photoshopelements, 2017. [Dostopano: 5. 2. 2017].

[2] Adobe Photoshop Lightroom. Dosegljivo: https://lightroom.adobe.com/, 2017. [Dostopano: 5. 2. 2017].

[3] OpenCV: Affine Transformations. Dosegljivo: http://docs.opencv.org/master/d4/d61/tutorial_warp_affine.html, 2017. [Dostopano: 28. 1. 2017].

[4] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen. Face description with local binary patterns: Application to face recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(12), December 2006.

[5] Android (operating system) — Wikipedia, The Free Encyclopedia. Dosegljivo: https://en.wikipedia.org/wiki/Android_(operating_system), 2016. [Dostopano: 8. 1. 2017].

[6] The Activity Lifecycle — Android Developers. Dosegljivo: https://developer.android.com/guide/components/activities/activity-lifecycle.html, 2017. [Dostopano: 23. 1. 2017].

[7] Services — Android Developers. Dosegljivo: https://developer.android.com/guide/components/services.html, 2017. [Dostopano: 31. 1. 2017].

[8] AsyncTask — Android Developers. Dosegljivo: https://developer.android.com/reference/android/os/AsyncTask.html, 2017. [Dostopano: 2. 2. 2017].

[9] Lélis Baggio. *Mastering OpenCV with Practical Computer Vision Projects*. Packt Publishing Ltd., 2012.

[10] Borut Batagelj. *Prepoznavanje človeških obrazov s pomočjo hibridnega sistema : doktorska disertacija*. PhD thesis, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2007.

[11] ART and Dalvik — Android Open Source Project. Dosegljivo: https://source.android.com/devices/tech/dalvik/, 2017. [Dostopano: 8. 1. 2017].

[12] Setup Eclipse (DEPRECATED) — Android Developers. Dosegljivo: https://developer.android.com/ndk/guides/setup-eclipse.html, 2017. [Dostopano: 28. 1. 2017].

[13] Eigenface - Wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Eigenface, 2017. [Dostopano: 5. 2. 2017].

[14] Eigenface Practical Implementation - Wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Eigenface#Practical_implementation, 2017. [Dostopano: 2. 2. 2017].

[15] Face Recognition. Dosegljivo: https://play.google.com/store/apps/details?id=com.seakleng.facedetection, 2017. [Dostopano: 5. 2. 2017].

[16] Face Recognition. Dosegljivo: https://play.google.com/store/apps/details?id=ch.zhaw.facerecognition, 2017. [Dostopano: 5. 2. 2017].

[17] Windows Hello — Windows 10 — Microsoft. Dosegljivo: https://www.microsoft.com/en-us/windows/windows-hello, 2017. [Dostopano: 5. 2. 2017].

[18] Facebook. Dosegljivo: https://www.facebook.com/, 2017. [Dostopano: 5. 2. 2017].

[19] Gabor wavelet - Wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Gabor_filter, 2017. [Dostopano: 9. 2. 2017].

[20] Google+. Dosegljivo: https://plus.google.com/, 2017. [Dostopano: 5. 2. 2017].

[21] Google Photos. Dosegljivo: https://www.google.com/photos/about, 2017. [Dostopano: 5. 2. 2017].

[22] PicFace Celebrity Matchup. https://play.google.com/store/apps/details?id=com.picitup.

[23] Celebrity Look Alike App Free. Dosegljivo: https://play.google.com/store/apps/details?id=com.face_compare, 2017. [Dostopano: 8. 2. 2017].

[24] Google Play. Dosegljivo: https://play.google.com/store/apps, 2017. [Dostopano: 23. 1. 2017].

[25] My Lookalike. Dosegljivo: https://play.google.com/store/apps/details?id=com.esense.android.gms.samples.vision.face.mylook, 2017. [Dostopano: 8. 2. 2017].

[26] My Twin Celeb. Dosegljivo: https://play.google.com/store/apps/details?id=com.picscout.mytwinceleb, 2017. [Dostopano: 8. 2. 2017].

[27] Howse. *Android Application Programming with OpenCV 3*. Packt Publishing Ltd., 2015.

[28] Aapo Hyvärinen and Erkki Oja. Independent Component Analysis: Algorithms and Applications. *Neural Networks Research Centre Helsinki University of Technology*, 2000.

[29] Java (programming language) — Wikipedia, The Free Encyclopedia. Dosegljivo: `https://en.wikipedia.org/wiki/Java_(programming_language)`, 2016. [Dostopano: 8. 1. 2017].

[30] Jernej Konda. Prepoznavanje starosti oseb s slik obrazov z uporabo konvolucijskih nevronskih mrež. Diplomska naloga, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2016.

[31] Linear discriminant analysis - Wikipedia. Dosegljivo: `https://en.wikipedia.org/wiki/Linear_discriminant_analysis`, 2017. [Dostopano: 9. 2. 2017].

[32] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *IEEE ICIP 2002*, 2002.

[33] Luxand Face Recognition. Dosegljivo: `https://play.google.com/store/apps/details?id=com.luxand.facerecognition`, 2017. [Dostopano: 5. 2. 2017].

[34] Math - The Commons Math User Guide - Linear Algebra. Dosegljivo: `http://commons.apache.org/proper/commons-math/userguide/linear.html`, 2017. [Dostopano: 29. 1. 2017].

[35] OpenCV — OpenCV. Dosegljivo: `http://opencv.org/`, 2017. [Dostopano: 23. 1. 2017].

[36] OpenCV - Bug #3565: Portrait orientation on Android - OpenCV DevZone. Dosegljivo: `http://code.opencv.org/issues/3565`, 2017. [Dostopano: 23. 1. 2017].

[37] OpenCV Manager – Aplikacije za Android v storitvi Google Play. Dosegljivo: https://play.google.com/store/apps/details?id=org.opencv.engine, 2017. [Dostopano: 31. 1. 2017].

[38] O. M. Parkhi, A. Vedaldi, and A. Zisserman. Deep face recognition. In *British Machine Vision Conference*, 2015.

[39] Florian Schroff , Dmitry Kalenichenko , James Philbin. Facenet: A unified embedding for face recognition and clustering. Technical report, Google Inc., 2015.

[40] macOS - Photos - Apple. Dosegljivo: http://www.apple.com/macos/photos/, 2017. [Dostopano: 5. 2. 2017].

[41] Picasa - Wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Picasa, 2017. [Dostopano: 5. 2. 2017].

[42] pictriev, face search engine. Dosegljivo: http://www.pictriev.com/, 2017. [Dostopano: 5. 2. 2017].

[43] FaceFirst Face Recognition Software. Dosegljivo: https://www.facefirst.com, 2017. [Dostopano: 5. 2. 2017].

[44] FaceTec — Face Recognition for the Real World. Dosegljivo: https://facetec.com, 2017. [Dostopano: 5. 2. 2017].

[45] Zhen Lei Lun Zhang Stan Z. Li Shengcai Liao, Xiangxin Zhu. Learning multi-scale block local binary patterns for face recognition. In *International Conference on Biometrics (ICB)*, 2007.

[46] Stack Overflow. Dosegljivo: http://stackoverflow.com/questions/19611062/camera-face-detection-getmaxnumdetectedfaces-returns-0-for-nexus-4-nexus-7-sam, 2017. [Dostopano: 23. 1. 2017].

[47] Eigenface Connection with SVD - Wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Eigenface#Connection_with_SVD, 2017. [Dostopano: 2. 2. 2017].

[48] Yaniv Taigman , Ming Yang , Marc'Aurelio Ranzato , Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. Technical report, Facebook AI Research, 2014.

[49] XML - Extensible Markup Language. Dosegljivo: https://www.w3.org/XML/, 2016. [Dostopano: 7. 1. 2017].